5th Annual multicoreEXPO

Applying Industry Standard Multicore Programming Practices to Migrate C/C++ Code

Scott A. Hissam

Carnegie Mellon Software Engineering Institute

ME735

Learn today. Design tomorrow.
ESC
Silicon Valley • April 26 - 29, 2010
McEnery Convention Center • San Jose

5th Annual multicoreEXPO

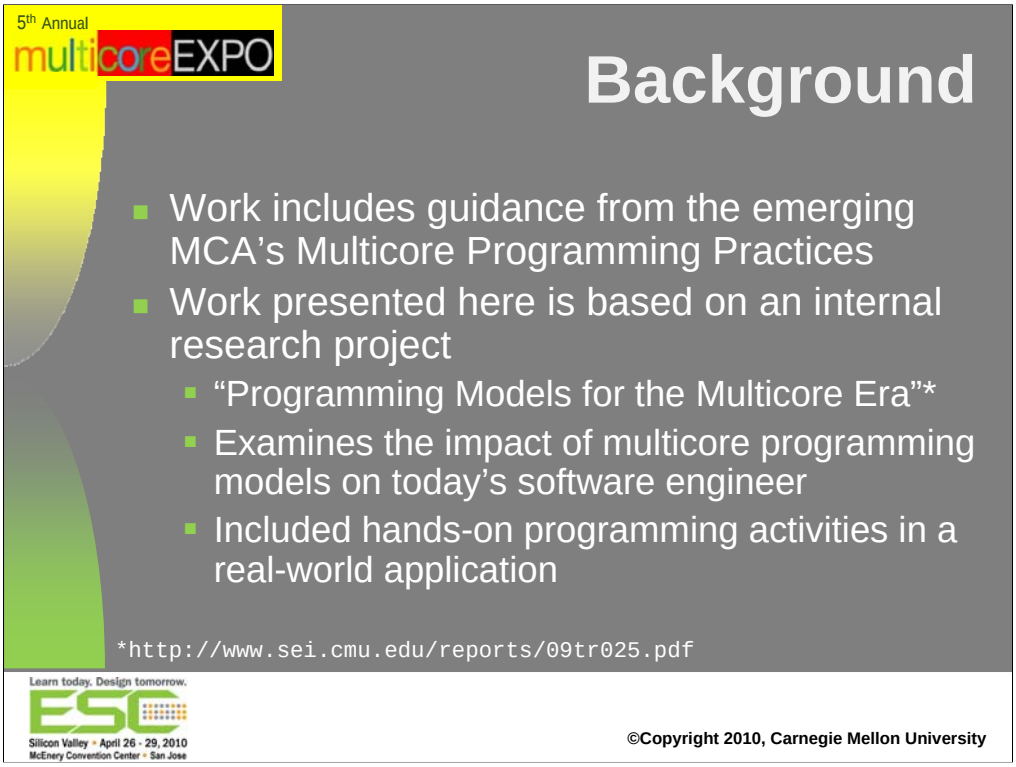
5th Annual multicoreEXPO

Agenda

- Background
- Face Recognition
- Approach
  - Prepare
  - Measure
  - Tune
  - Repeat
- Lessons

Special Thanks to:

- Kurt Wallnau
- Dan Plakosh
- James Ivers



You can learn more about the Multicore Association's Multicore Programming Practices at http://www.multicore-association.org/workgroup/mpp.php

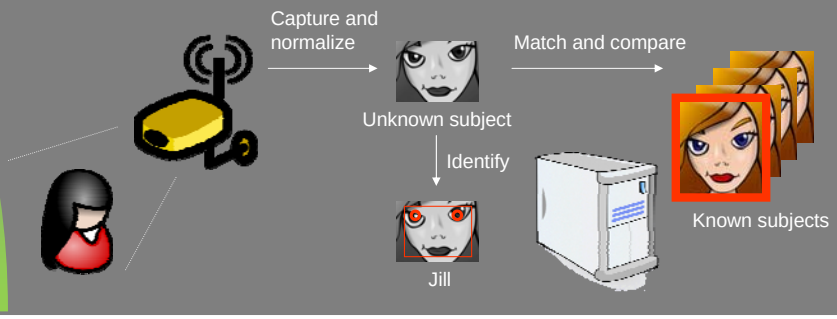
You can learn more about the SEI's IRAD on Multicore Programming Models at http://www.sei.cmu.edu/reports/09tr025.pdf

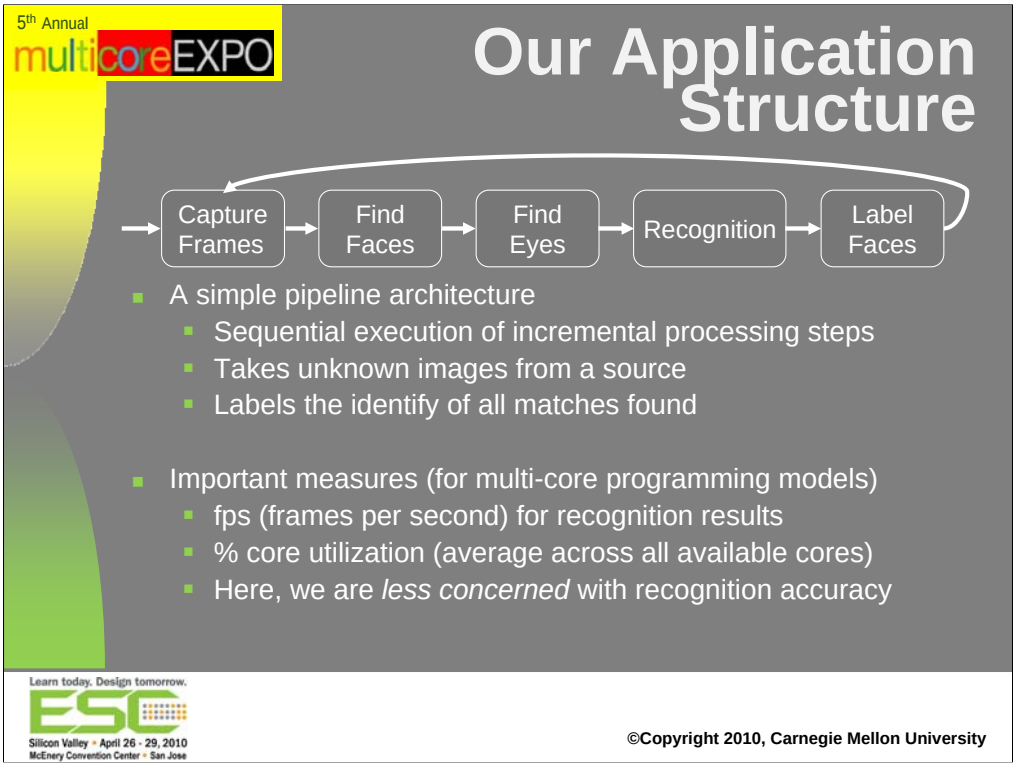
Face Recognition

- Goal is not to reinvent, but
  - Integrate legacy code to use multicore
  - Understand issues that arise using different methods
  - Apply our own “lessons”
- Why This Application?
  - Computationally demanding for graphics and databases
  - Easy to observe and measure quality of service improvements
  - Legacy rich!

Face Recognition

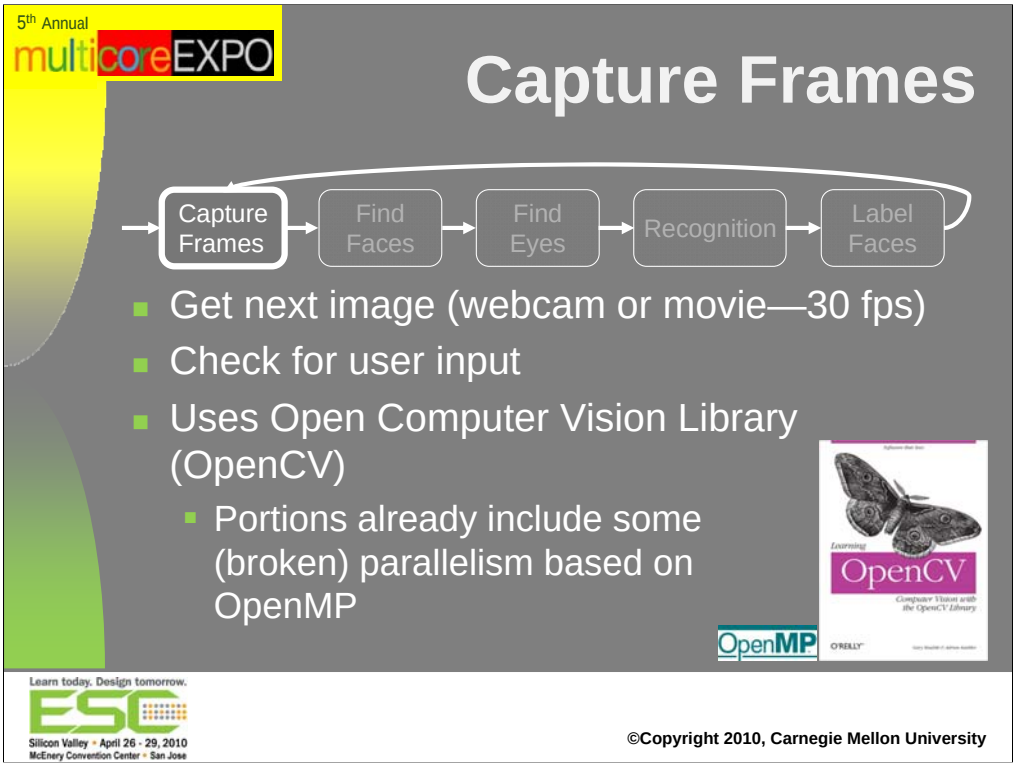
- Face recognition is an image processing application that attempts to identify an unknown subject from a set of known subjects, each of which is associated with a set of known images





Ultimately, the best measure of face recognition performance in many applications will vary. In some, frames per second could in fact trump accuracy (i.e., counting people in queues or in a store) where as in other applications frames per second have absolutely no bearing on the ability to recognize a face (searching a DB of criminals against suspects) but the measure of merit is accuracy.

For our application in investigating multicore programming models, frames per second and cpu utilization are important.



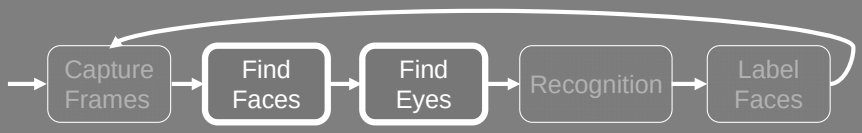
There is a nice history of OpenCV (http://sourceforge.net/projects/opencvlibrary/) at http://www.willowgarage.com/pages/software/opencv.

There is a nice history of OpenMP (http://openmp.org/wp/) at https://computing.llnl.gov/tutorials/openMP/#History

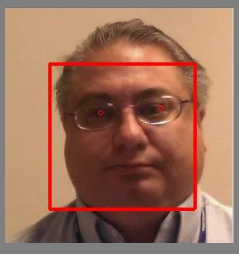
Bradski, G., Kaehler, A., Learning OpenCV: Computer Vision with the OpenCV Library, O'Reilly Press, ISBN: 978-0596516130

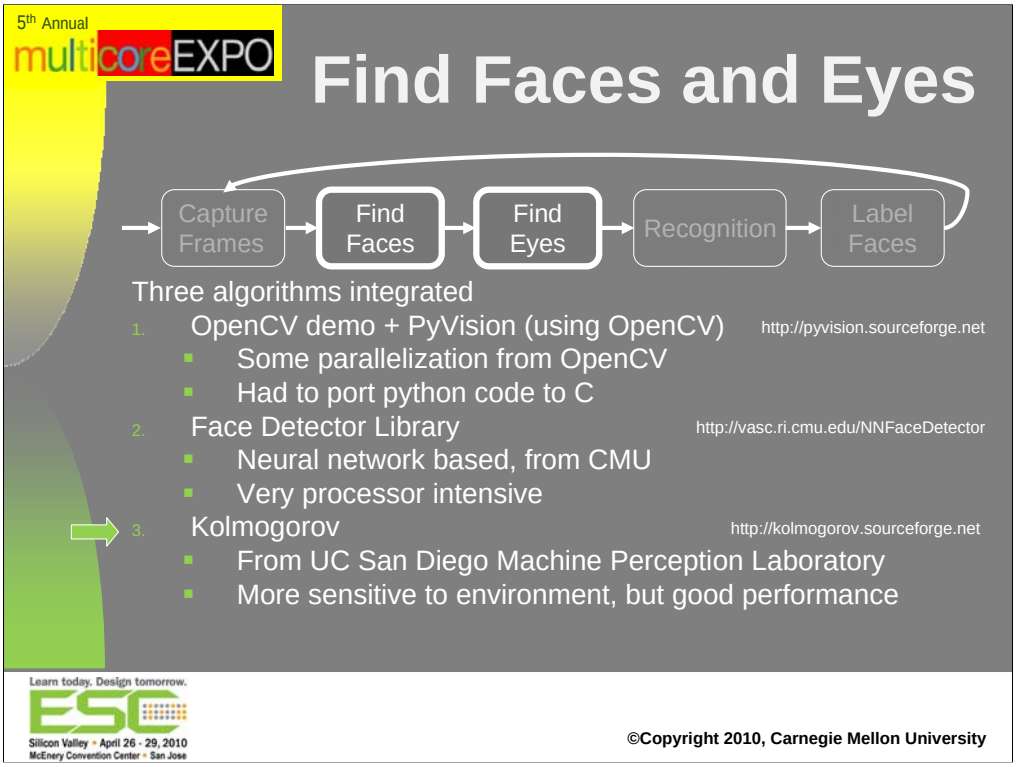
Mattson, T., Sanders, B., Massingill, B., Patterns for Parallel Programming, Addison-Wesley Professional, ISBN: 978-0-321-22811-6

Find Faces and Eyes



- Find regions within image that *could* contain a face
- Find eyes within each region
- Various filters applied, depending on algorithms used
  - Rescale to uniform size
  - Convert to black and white
  - Center based on eye location
  - Normalize lighting





Our application was able (at compile time) to select between three algorithms for Finding Faces and Eyes.

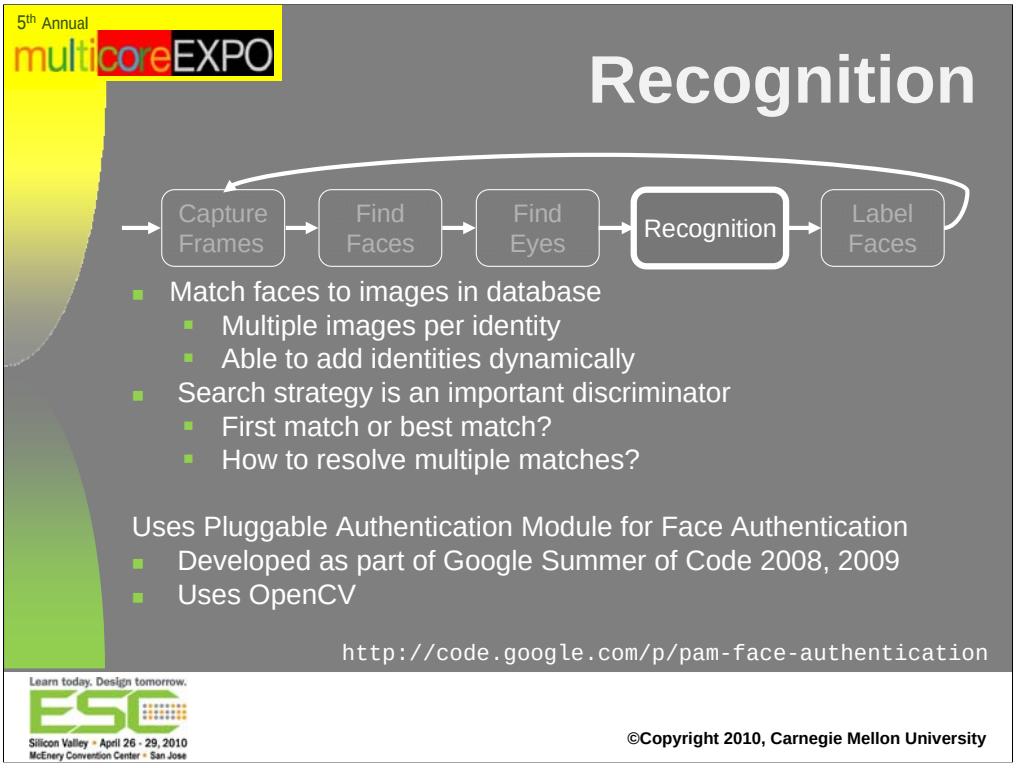
For the purpose of our experiment, we settled on the Kolmogorov Project (kolmogorov.sourceforge.net and http://freshmeat.net/projects/kolmogorov/) supporting fast face detection in images using MPISearch (http://mplab.ucsd.edu/grants/project1/free-software/mpi-search.html).

The eye detection portion was originally written in python from PyVision (pyvision.sourceforge.net) which performs the eye detection algorithm and calls OpenCV via python hooks into OpenCV. Although an adequate approach, adding this needed algorithm using an additional level of indirection would result in complicating our multicore programming model investigation in that:

-The python runtime would either have to be integrated into our main processes memory space or fork/spawned as another OS-level process

-Measurement and analysis of the runtime for parallelization would be masked by the python runtime (integrated or as a separate process—neither of which would be changed)

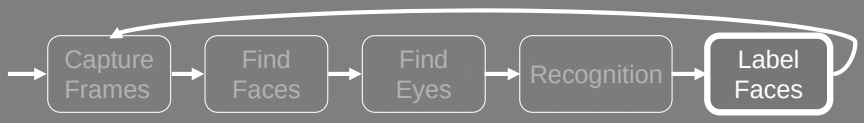
-The additional level of indirection was simply no needed as our application was already using OpenCV



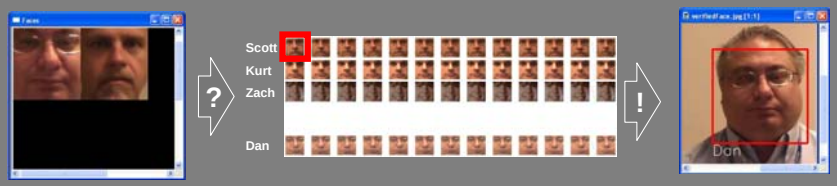
To increase “accuracy” of face detection, the approach used in our application is to match the unknown subject with multiple images from known subjects (hopefully from various angles) to increase the likelihood of positive matches (thus reducing false matches).

Additionally, to increase the likelihood of a positive match, the search can take the first match (which may or may not be positive) or continue to search and match to get more “hits”. More hits can help resolve multiple matches of the unknown subject to the known subjects, but collisions are still possible. Out approach was to use First match (recall accuracy is not a key quality attribute of our research).

Label Faces



- Simple: for each face, if a matching identity is found, label the image



5th Annual multicoreEXPO

Agenda

- Background
- Face Recognition
- Approach
  - Prepare
  - Measure
  - Tune
  - Repeat
- Lessons

5th Annual multicoreEXPO

Approach

- Prepare
- Measure
- Tune
- Assess
- Repeat…

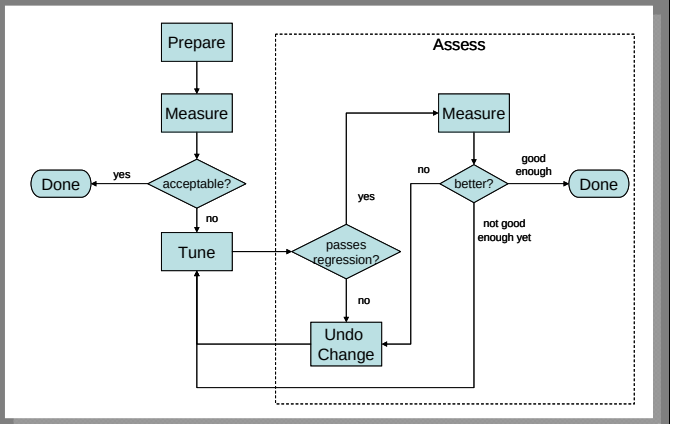


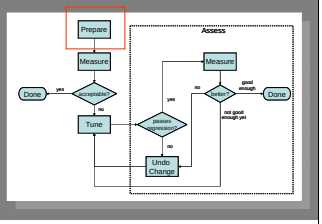
Figure borrowed from the emerging Multicore Programming Practices Guide, Analysis and High Level Design

Key to effective performance tuning is discipline. Use measurements and careful analysis to guide decision making, change one thing at a time, and meticulously measure again to confirm that changes have been beneficial. The idea is to use an iterative approach based on the steps shown in figure. Such an iterative approach can be used regardless if one is tuning a serial or parallel application.

5th Annual multicoreEXPO

Prepare

- Assemble a collection of regression tests and benchmarks against which to improve the program's performance



In preparing ask yourself, What are your performance goals? What performance metrics make sense? What use scenarios are most likely to benefit from improvements?

Preparing OpenCV

Early benchmarks uncovered a bug

- On multicore machines, “find faces” resulted in numerous false positives
  - Where as single core machines fared better
- Now… find the bug…
  - Multicore, race conditions, unprotected shared access, OpenMP, oh my!
- Use of OpenMP itself was a hint, as it has the ability for detecting multiple CPUs, **`omp_get_num_procs()`**
- The bug was traced to **`cvSeqPushMulti()`** which is used only when # of CPUs (i.e., **`max_threads`**) **`> 1`**
  - Forcing **`max_threads`** to **`1`** confirmed this

`http://tech.groups.yahoo.com/group/OpenCV/message/61048`

Before the repair, the version of cvhaar.cpp tested on an 8 core Xeon machine running Windows XP, resulted in a total face detection time of 2340 milliseconds identifying 22 faces, of which 7 were not faces at all.

After the repair, the same image on the same machine was tested resulting in the accurate detection of 15 faces in only 95 milliseconds (a 2 orders of magnitude increase in performance). This speed up is well beyond anyone's expectations (if the original speed of 2334 milliseconds were to be improved by 8x, then one should expect the resulting speed to be on the order of a 1 magnitude increase in performance). Therefore the problem was that the work being performed was likely being duplicated.

In and of itself, the identification of false faces could be simply considered a nuisance, but in the architecture of our application, that meant that search and identification effort would be expended to find faces that will never be found! That is not a nuisance—that is a problem.

5th Annual multicoreEXPO

Lesson #1 Multicore Shall Do No Harm

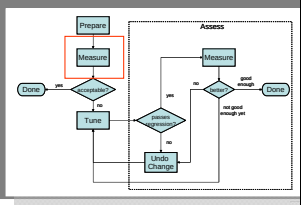
- False positives resulted in ***more*** time in places
  - Speed-down!
- Simple regression tests would have uncovered this early
- General purpose tools can not assume use

	orig cvhaar.cpp		new cvhaar.cpp	
image	1 thread	2 threads	1 thread	2 threads
P7250034	faces=10	faces=22	faces=10	faces=10
	ms=1881.73	ms=1210.35	ms=1894.79	ms= 973.42
dom1nationII012	faces= 7	faces=14	faces= 7	faces= 7
	ms=1107.90	ms= 726.84	ms=1107.33	ms= 572.02
crowd	faces= 4	faces= 7	faces= 4	faces= 4
	ms= 160.58	ms= 98.50	ms= 161.70	ms= 84.13
barca2_fd	faces=16	faces=23	faces=16	faces=16
	ms= 474.63	ms=1899.87	ms= 474.74	ms= 245.54
barca2	faces=15	faces=21	faces=15	faces=15
	ms= 475.85	ms=2022.23	ms= 479.60	ms= 247.38

5th Annual multicoreEXPO

Measure

- Gather performance data through profiling or simulation.
  - If the performance is reasonable, stop.
  - Of course, it probably won't be on the first iteration.



Various means can be used to effectively measure an take a performance profile of an application. One approach is to add performance counters or instrumentation directly into the code, another is to use an external tool to observe and record the performance of the application from counters available in the execution environment.

Simulation can also be used to emulate (or simulate) the execution of the application absent the actual execution environment… but relative speedup achieve in code or software architecture improvements can often map to the actual hardware environment (assuming the fidelity of the simulator or emulator).

Processing Time / Module

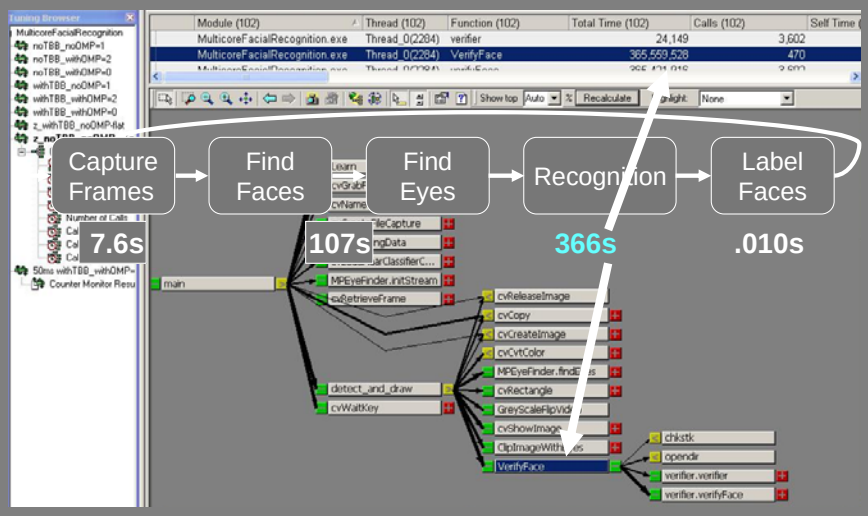
1.56% **22.17%** **75.61%** **0.00%**

- Execute the sequential version of the application and use a profiler to understand where time is spent
  - OpenMP parallelization in OpenCV can be "turned off"
  - Used Intel's VTune Performance Analyzer to gather data
- The "hotspot" appears to be in Recognition (at a high level)
  - Consumes most processing time
  - Amenable to multiple parallelization strategies

*Modules account for 99.33% of total processing time

5th Annual multicoreEXPO

HotSpot Analysis

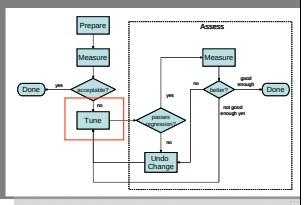


Learn today. Design tomorrow.
ESC Silicon Valley • April 26 - 29, 2010
McEnery Convention Center • San Jose

5th Annual multicoreEXPO

Tune

- Look at the measurements
  - Understand where the program is spending time and whether this is reasonable.
  - Find the biggest bottleneck that you can improve and make a single change



Take the time to understand what data the profiler or the simulators is providing you. Often, misinterpretation of the data can lead to missteps and focusing effort “chasing cycles”. What time is computing (actual processor execution) and what time is waiting (blocking)?

Assess

- Determine if the change was positive.
  - This means that no bugs were introduced and that performance improved.
  - If the change was negative, then you need to back out the change and improve your understanding before trying again

Improvement Strategies

- Tuning vs. re-architecting: a matter of perspective
  - Many of the same tools (e.g., changes in problem decomposition) can be used in more or less invasive ways.
  - Minimal, local changes may have more or less impact than more drastic, global changes—you have to look at the data.
  - What can be efficiently changed depends on many factors, such as
    - Access to/knowledge of source code (e.g., legacy code)
    - Willingness to alter critical or difficult to understand code
    - Ripple effects due to dependencies and coupling
- Lesson #2: There is still a great deal of *art* in effectively tuning applications, even in the sequential world.

Basic Parallelization Strategies

Decompose a problem into segments (of data) and apply a function in parallel.

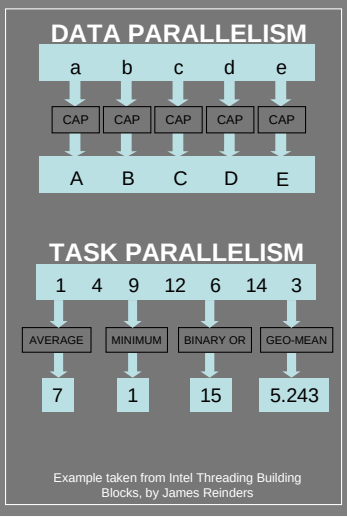
- Generally used to process a unit of computation in a shorter period of time
- Data parallelism

Decompose a problem into distinct tasks that share data.

- Generally used to do more units of work over a period of time
- Task or function parallelism

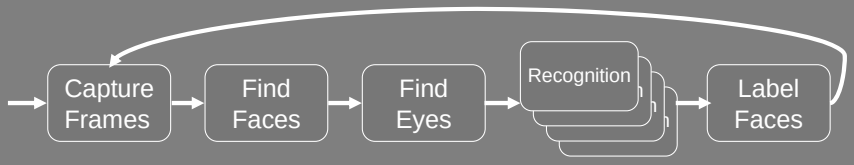
Use a combinations of strategies.

- Multi-level or multigrain parallelism



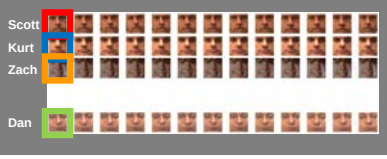
Example taken from Intel Threading Building Blocks, by James Reinders

Parallelization Changes—Tuning

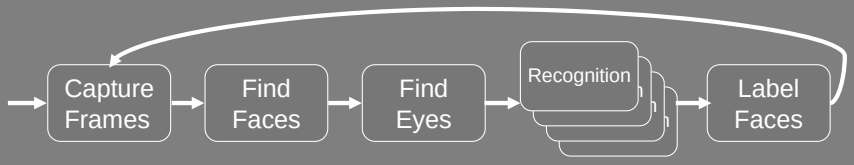


Focus on the “hotspot”—Recognition

- Use task parallelism to search for matches against multiple identities concurrently.
- Enable data parallelism in OpenCV.



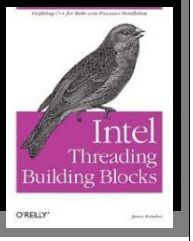
Parallelization Changes—Tuning



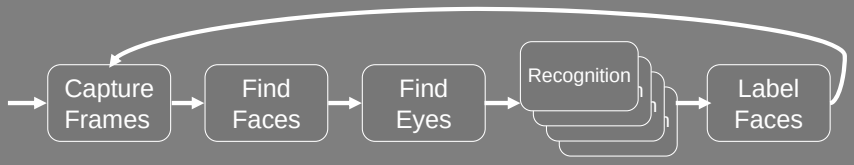
Focus on the "hotspot"—Recognition

- Use task parallelism to search for matches against multiple identities concurrently.
- Enable data parallelism in OpenCV.

Threading Building Blocks (TBB)
• C++ template library from Intel
• Supports task and data parallelism
• Consists of algorithms and data structures
• Requires code to be restructured



Parallelization Changes—Tuning



Focus on the "hotspot"—Recognition

- Use task parallelism to search for matches against multiple identities concurrently.
- Enable data parallelism in OpenCV.

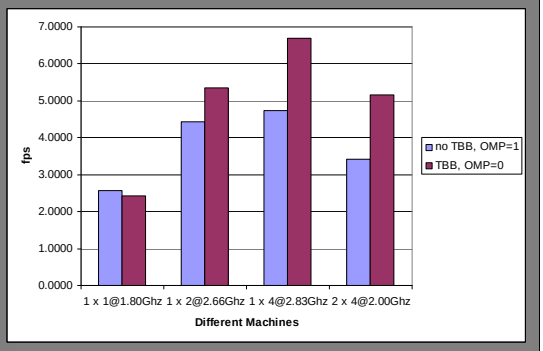
OpenMP
• API for shared memory multiprocessing in C/C++ and Fortran
• Owned by a group of hardware and software vendors
• Primarily for data parallelism
• Does *not often* require code to be heavily restructured

OpenMP

Tuning Results—1

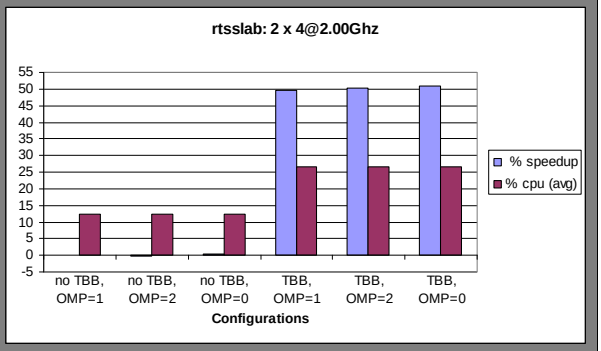
Observations

- Tuning improves fps on *most* machines.
- The 8 core machine doesn't perform best!
- fps seems to improve with clock speed.



- Analysis shows that cores are not kept busy, so of course clock speed is more significant—additional improvements should change the picture.
- Lesson #3: For some problems, however, this may be a realistic outcome.

Tuning Results—2



- OpenMP (data parallelism) had little impact
  - Local improvements don't necessarily solve global problems
- TBB (task parallelism—here a global change) had a much larger impact
  - But why isn't it closer to 8x faster?

Parallelization Limits

- Amdahl's Law expresses the maximum expected improvement when one portion of a program is improved.
  - Attempting to parallelize recognition (75.6% of computations, *P*) across 8 (*N*) cores, the theoretical maximum speedup would be a factor of 2.95 (10.12 fps)
  - At this point we have a factor of 1.51 (5.17 fps)

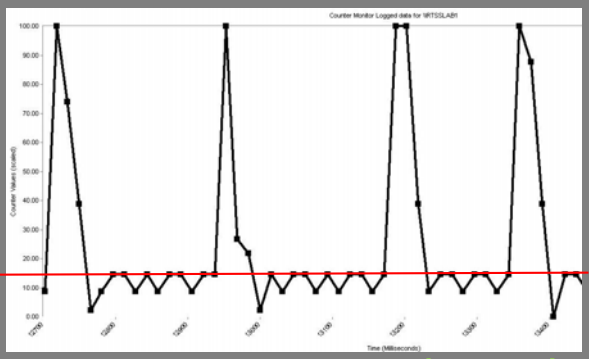
$$\frac{1}{(1-P)+\frac{P}{N}}$$

P is the proportion that can be parallel

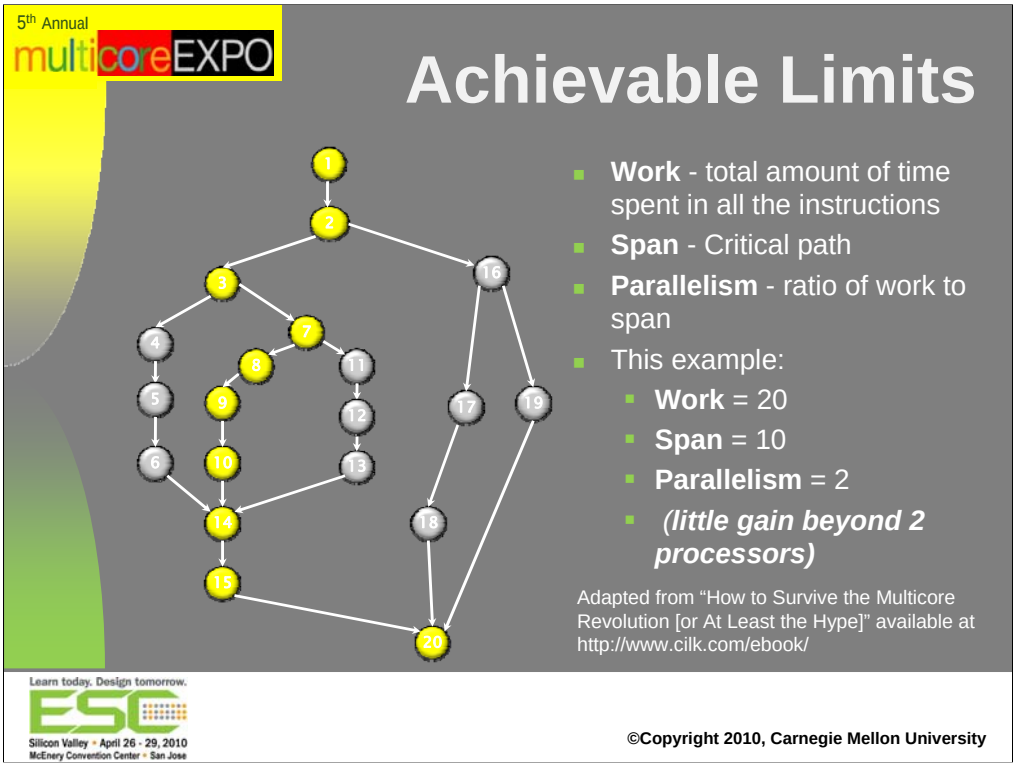
N is the number of processors

Detailed Look at Utilization

12.5% is where one core is kept busy (sequential behavior)



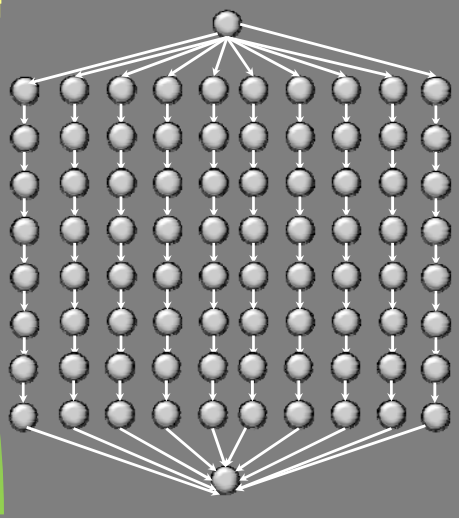
Several frames, with a recognizable pattern



Leiserson, C., Mirman, I., "How to Survive the Multicore Software Revolution (or at Least Survive the Hype)", http://software.intel.com/en-us/articles/e-book-on-multicore-programming/

5th Annual multicoreEXPO

Limits in the Ideal



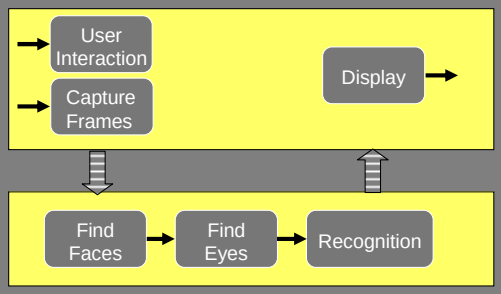
- The more computations that can parallelized the bigger the advantage
- This example:
  - **Work** = 82
  - **Span** = 10
  - **Parallelism** = 8.2

Adapted from "How to Survive the Multicore Revolution [or At Least the Hype]" available at http://www.cilk.com/ebook/

Parallelization Changes: Re-architecting

De-couple real-time from softer functionality

- Guarantees at real-time level improve responsiveness and allow new functionality
- More options for "fall behind" strategies at softer level



- A key implementation feature needed for this change is thread affinity
  - Restricts activities to a set of cores
  - *Not available* in current OpenMP standard; no extensions in our compiler (though extensions do exist in Intel's compiler)

"Data and Thread Affinity in OpenMP Programs." Terboven, an Mey, Schmidl, Jin, and Reichstein. 2008 Workshop on Memory Access on Future Processors: a Solved Problem? May 5, 2008.

5th Annual multicoreEXPO

Platform for Future Research

- Many different topics can be explored in this context, such as
  - Application of additional parallelization strategies, fine or coarse grained
  - Code structures that facilitate/hinder analysis and parallelization
  - Interference issues among programming models
  - Appropriate strategies for employing multiple levels of parallelism
  - Migration strategies for multi-core
    - What architectures are appropriate for different applications and their driving quality attributes?
    - What features does industry need to supply?
    - How can we migrates applications as technologies evolve?

5th Annual multicoreEXPO

Lessons

- Multicore Shall Do No Harm
  - Know the trajectory changes are making!
  - Measure, assess, repeat
- Tuning is still an art, even in the sequential world
  - Local v. Global optimizations
  - Code-centric v. architecture-centric
- Know when to stop
  - For some problems, a realistic outcome may be less than the ideal
  - Set expectations accordingly

5th Annual multicoreEXPO

Contact Information

Scott A. Hissam
Carnegie Mellon
Software Engineering Institute
Pittsburgh, PA 15213 US
shissam@sei.cmu.edu
+1.412.268.6526

5th Annual multicoreEXPO

Backups

5th Annual multicoreEXPO

Abstract

- Migrating a sequential C/C++ application to a multicore platform can be a daunting task, often requiring software engineers to modify many aspects of the application such as its high level structure, software code, and data organization to make it multicore ready. The MCA's Multicore Programming Practices (MPP) provides practical guidance for the migration of C/C++ application to a multicore platform. This talk shares insights from our experience using MPP tenets during the engineering of a facial recognition system. It includes thoughts on getting the code right for both single- and multicore, using measurement to find and improve hotspots, using architectural patterns to make both naive and more effective improvements, and using available multicore programming models--problems and all.

5th Annual multicoreEXPO

About the Speaker

- Scott A. Hissam is a senior member of the technical staff for the Carnegie Mellon Software Engineering Institute where he conducts research on component-based software engineering, open source software, and multicore. Scott just completed an internal research effort on “Programming Models for the Multicore” which sought to understand current and emerging multi-core programming models and to identify trends and gaps in those models as it would pertain to mainstream software engineers. His publications include two books, Building Systems from Commercial Components and Perspectives on Free and Open Source Software, papers published in international journals, and numerous technical reports. He has a Bachelor of Science degree in Computer Science from West Virginia University.

Root of All Evil

"Programmers waste enormous amounts of time thinking about, or worrying about, the speed of noncritical parts of their programs, and these attempts at efficiency actually have a strong negative impact when debugging and maintenance are considered. We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil"

--Donald Knuth. "Structured Programming with go to Statements." Computing Surveys, vol 6, no 4, Dec 1974, p 268